1

BATTLE MANAGEMENT VISUALIZATION SYSTEM

THESIS

Charles Lewis Wardin
Captain, USAF

AFIT/GE/ENG/89D-56

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

(1)

AFIT/GE/ENG/89D-56

BATTLE MANAGEMENT VISUALIZATION SYSTEM

THESIS

Charles Lewis Wardin
Captain, USAF

AFIT/GE/ENG/89D-56

AFIT/GE/ENG/89D-56

BATTLE MANAGEMENT VISUALIZATION SYSTEM

THESIS

Presented to the Facu.ty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Charles Lewis Wardin, B.S.E.E.
Captain, USAF

December, 1989

Acknowledgments

I would like to thank all those who have given me support during this thesis effort. First I want to thank Major Phil Amburn, my thesis advisor and so much more. His technical expertise combined with his easy going nature made him the ideal advisor for me. Thanks also to all the other graphics students who helped me find those hidden programming bugs. Thanks to these students and Major Amburn for putting up with my endless barrage of sarcasm. I would like to thank my sponsors, Armstrong Aerospace Medical Research Laboratory, National Oceanic and Atmospheric Administration and the Air Force Office of Scientific Research for their support.

Most of all I thank my wife and family. Kids, you helped me keep things in perspective; to know when to put the school work down and throw the football with you. Teena, this just wouldn't have happened without your love and support. With this in mind, I dedicate this report to you with love.

Charles Lewis Wardin

Table of Contents

List of Figures

List of Tables

AFIT/GE/ENG/89D-56

Abstract

This investigation deals with the development of a software system to explore the concept of previewing Air Tasking Orders in a three-dimensional virtual environment. The virtual environment is created by displaying three-dimensional graphics images onto a graphics workstation screen or onto a head-mounted display. The software system, entitled the Battle Management Visualization System (BMVS), supports a variety of input devices to suit different situations. The user is presented with a miniature battle environment complete with aircraft, threat regions, targets, etc. The aircraft *fly* along paths and arrive over the targets at the simulated time over target time. The user may also *fly* along the path to create a plane's eye view. The user can *move* about the environment to examine various aspects close up or zoom out for an overall view. The results of the effort support the idea that a tool to preview ATOs is feasible.

BATTLE MANAGEMENT VISUALIZATION SYSTEM

I. Introduction

This thesis deals with the development of a software system to allow a person to preview what the execution of an Air Tasking Order may look like. The system will support a variety of input and output devices to allow the individual user to tailor the system to his or her tastes. In addition, the effectiveness of *virtual world* generated by a head-mounted display will be explored. Finally the applicability of this software system for non-combat operations such as FAA skyway management will be investigated.

1.1 Background

Advances in tactical battlefield technology provide the battle manager with more information than ever before. The battle manager uses the information to make critical mission decisions. However, the amount of information presented may exceed a human's information handling ability (17:619). It is clearly an undesirable situation when the battle manager cannot absorb all the needed facts to make the best decision.

The decisions the battle manager makes, are communicated to the lower echelons via an Air Tasking Order (ATO). The ATO specifies (among other things) the resources to be used and the targets selected for *tomorrow's war*. Today's battle manager has limited ways to evaluate the quality of the ATO (and therefore his decisions). He may draw upon the collective experience of his staff or wait for the ATO to be executed and review the results. Both methods of evaluation have drawbacks. ATOs are complex documents allocating hundreds of resources. Even an experienced staff may miss the *holes* or inconsistencies within an ATO. Post ATO execution review may aid the battle manager with future ATOs. However valuable this hindsight is, it is just that - hindsight.

What a great help it would be to the battle manager to give him a crystal ball and let him see the results of his ATO. This would allow him to optimize the ATO for the most damage to the enemy with the fewest losses of his resources. He would gain all of the needed information very quickly as well; after all he need only look at the picture the crystal ball provides. He not only produces the best plan possible, he produces it without reaching his information handling limit.

While the BMVS does not promise to be that magical crystal ball, perhaps it can provide the battle manager with some additional timely information to achieve some of the same benefits. The BMVS will provide the battle manager with a three dimensional perspective view of the battle area. The 3-D view allows the battle manager to instantly recognize fiercely defended areas (by the threat zones being shown). The defense information is available, if somewhat embedded, in the ATO but it is certainly not as quickly found. Figure 1 compares an example of a tabular representation of threat zones, a 2-D map version representing threat zones, and a 3-D representation of threat zones.

While the use of a BMVS in an operational situation as described above shows promise, using a BMVS in a training situation may show more. A system such as the BMVS could be integrated into a training environment without disrupting operational missions. An inexperienced battle manager or inexperienced battle staff members could learn by creating various ATOs and then previewing the ATOs and get a *feel* for how the contents of an ATO might be executed.

1.2 Problem Statement

A software system can be designed to provide a battle manager with a preview of where his resources will be during ATO execution.

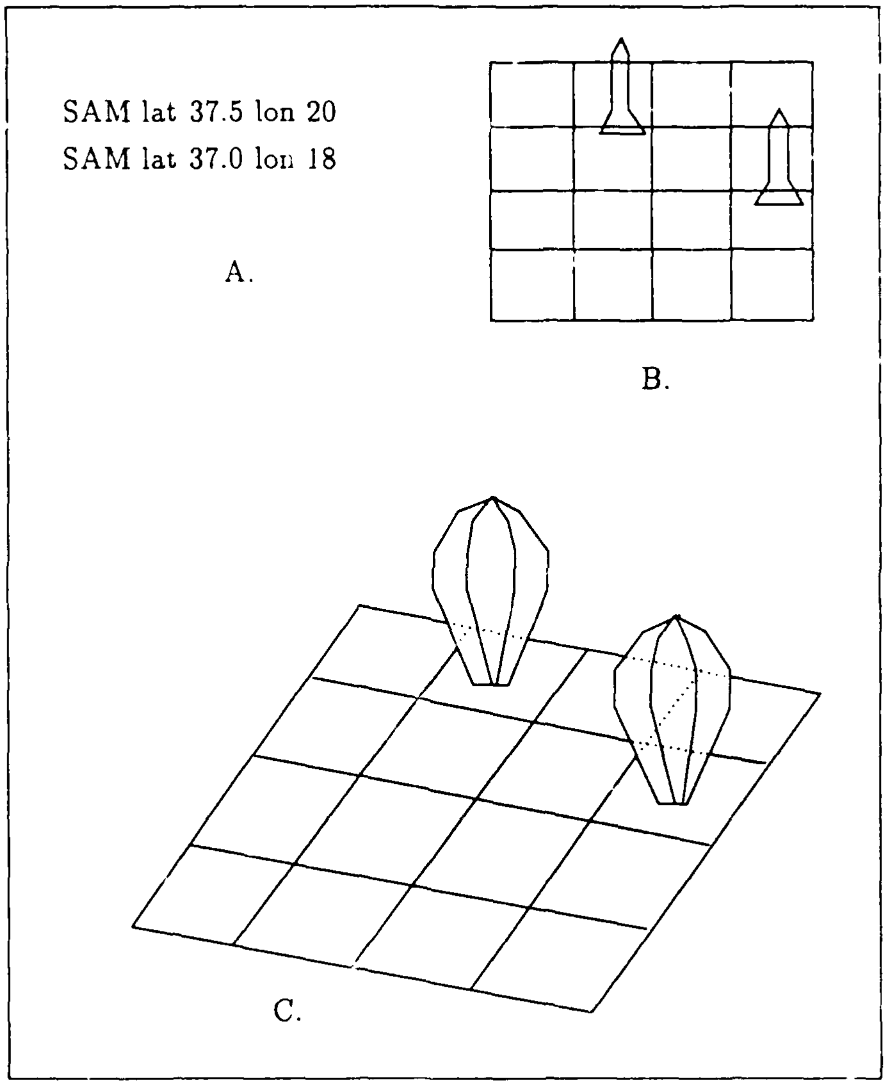

Figure 1. Tabular (A.), 2D (B.), and 3D (C.) representations of threat zones

1.3 Research Questions

The following questions are the objects of this research:

1. Can effective use of computer graphics provide the battle manager useful previews of the execution of Air Tasking Orders (ATOs)?
2. Can a three dimensional *virtual world* (such as provided by a head mounted display) provide a natural man/machine interface for battle planning?
3. How can this approach aid in other areas, specifically non-combat applications such as FAA *Skyways* and Terminal Control Areas management.

1.4 Definitions

Battle Management The process of planning, tasking, deploying, directing, or controlling combat and support forces (2:66).

Computer Graphics The creation, storage, manipulation, and display of mathematical models or digitized pictures using a computer (5:3).

1.5 Assumptions

Several assumptions help *draw the road map* for the project. One, the users of the system have normal color vision. Two, the end user would have a computer dedicated to this task (that is, the computer would not be running several other tasks at the same time). Three, the graphic images need not be *photo-realistic* but should communicate well.

1.6 Scope

Certainly the use of computer graphics is not the only possible method to improve the ATO production process. It would be possible to incorporate some Artificial Intelligence into the system to aid the battle manager in interpreting the displayed information. However, this research is limited to the display issues involved with ATO preview.

Change is often met with resistance. This is no less true in the Air Force than in the civilian community. People in training however, tend to be more receptive toward change than the *old pros*. Therefore, the the system developed during the research is targeted for the training environment rather than the operational environment.

Focusing on the training environment also helps keep the research unclassified. Every effort is used to keep the all portions of the research (that is, software, documentation, and thesis) unclassified. This is accomplished by using generic (*dreamed up*) data for the theater, threat regions, etc. rather than real world data.

The work is deamed complete when a designed and implemented prototype software system exists that demonstrates the use of computer graphics for ATO preview. Specifically the BMVS completion criteria is as follows:

- It supports display on a standard CRT and the AFIT Head-Mounted Display.
- The frame rate is at an acceptable level (see section 3.1.2.2).
- It supports at least three input devices in various combinations.
- Portions of th virtual environment may be manipulated (edited) by the user.

II. Literature Review

Three interrelated factors create one inescapable problem of *information overload* for a battle manager. The first contributing factor is the mobility of today's enemy forces. The second factor is the sheer number of enemy forces. A third factor is the vast amount of data that computer-based sensors collect on the enemy forces. Together these factors produce an immense amount of information that may go beyond the human's information handling ability (17:619). How can the battle manager make the best decisions when so overwhelmed with information?

Such an information overload affects all of the battle manager's functions (that is, planning, tasking, deploying, directing, and controlling). For example, one very specific problem involves the production of Air Tasking Orders (ATOs). ATOs consist of two parts. Part one contains high level guidance from the commander and part two contains detailed mission guidance from the commander's staff. Before an ATO can be produced, the staff must analyze information such as enemy activities, aircraft availability, etc. In addition, much of the ATO production is done manually. Therefore, the ATOs that reach the units may be based on data and guidance that is over 24 hours old. Once the unit personnel receive the ATO, they prepare mission sequence worksheets for each position and may post manual display boards (15:Sec 2, 40-44).

This chapter will discuss some current systems designed to ease the battle manager's workload. Also discussed is the potential of computer graphics for comunicating the needed information to the battle manager effectively. Finally, the concept of a virtual environment and its applicability to battle management is addressed.

2.1 Current Battle Management Aids (15:Apndx a, 1-9)

Several systems currently exist to help the battle manager and battle staff. The majority of these tools are Artificial Intelligence based decision aids. The following section describes some of the tools available and what benefits they bring to the battle staff.

2.1.1 Target Prioritization Aid (TPA). This tool is designed to prioritize targets in such a way that the enemie's sortie capability is minimized for a given number of friendly sorties. TPA outputs a list of prioritized targets that can be used by the combat planners. There is no graphical output generated by the TPA.

2.1.2 Cost/Benefit of Tactical Air Operations (CBTAO). The CBTAO system uses threat projection and friendly status information to predict the probability of success for nominated targets. The tool is a rule-based expert system and can incorporate actual battle results in order to remain accurate as battle conditions change. The CBTAO system also has the capability of explaining how the probability estimates were derived.

2.1.3 Command and Control Warfare Strategy Planning Aid (CTA). This tool allows the user to input potential attack options and receive numerical values for cost and benefit levels for each option. CTA is an Artificial Intelligence based aid that can be thought of as an automated checklist/spreadsheet for the battle planner.

2.1.4 Route Planning Aid (RPA). This tool is designed to aid individual flying units in planning the best flight path. RPA is a rule-based expert system that uses Sam-threat, terrain, and flight profiles to determine the least-lethal route. In addition, the system can provide the rationale for the route selection. RPA has graphical output capability but uses a standard alpha-numeric terminal for input.

2.1.5 Computer Assisted Force Management System (CAFMS) (15:Sec 2, 5) This system is the primary tool used tc build, update and disseminate ATOs. The CAFMS is a conventional processing system (i.e., not Artificial Intelligence based) which supports a variety of ADP file and distribution functions. No graphical output is produced by the CAFMS.

2.1.6 Tactical Expert Mission Planner (TEMPLAR) (6:Sec 2, 1) Of the tools discussed, TEMPLAR is the most comprehensive attempt to aid in the mission planning process. TEMPLAR uses knowledge-based artificial intelligence techniques to assist in ATO production. The tool checks completeness and soundness of mission profiles as well as the consistency of an evolving plan. In addition, TEMPLAR provides the planers with various options, ranking the possible solutions, and allowing *what if* type trade-offs. TEMPLAR supports both tabular output and a map-like graphical representation of the battle area.

2.2 Graphics as a Solution

As shown above, the problems facing the battle manager are well known and several systems have been developed to address the problems. However, most of the systems concentrate on providing some form of Artificial Intelligence technique for descision support. An alternative or supplemental approach is to communicate information to the battle manager with of computer graphics.

Could the use of computer graphics indeed help ease the battle manager's problems? The answer is an unqualified yes. Two years ago Capt Mark Kanko asked if computer graphics could help the pilot. Capt Kanko demonstrated that a three-dimensional model of a flight through hostile territory could be generated and displayed in less than one hour. He suggested as well that once the model was generated the flight could be played back in real-time when hosted on a fast computer (9:202).

The U.S. National Transportation Safety Board uses computer graphics to analyze aircraft accidents. Raw data from the flight recorders is difficult to analyze manually. The board's solution is to use the recorded data to generate graphic images of the plane. The images are then recorded onto video tape for real-time playback. The resulting animated sequence of the plane crash is easier to interpret than columns and columns of numbers (8:2).

Researchers at the Naval Post Graduate school recognized the battle manager's problem with information volume. Their proposed solution is a graphics based Command and Control Workstation. The researchers discuss several computer graphics methods to aid in the information transfer. One, the use of color in the display can be used to encode information. Two, the use of *windows* (or virtual displays) allows the user to tailor the display to his personal preferences. Three, the use of three-dimensional graphics (models drawn in perspective) can quickly show the commander the spatial relationships between objects (7:1-5).

2.3 A Virtual Environment

If the three-dimensional perspective view transmits information more effectively than a two-dimensional view, then a truly three-dimensional display could be the most effective. The display could simulate an environment to walk around in, point to situations of interest, and perhaps even *grab* and reposition virtual aircraft. We live in a three-dimensional world, so the information transfer would be quicker and easier using a three-dimensional virtual world (7:5). Large real-time holographic displays of the virtual world are beyond today's technology (7:5). However, there is an alternative to holograms.

One method of providing a virtual environment that is within today's technology is the Head-Mounted Display (HMD). A HMD presents the user with a three dimensional perspective view that changes as he moves his head. A working HMD was demonstrated over twenty years ago by Ivan Sutherland (14:757-764). Since that time many researchers have produced their own versions of HMDs. For example, NASA's Ames Research Center has produced a HMD (12:20-21) as has the Air Force Institute of Technology(AFIT) (10:6-8). Today there is even an off-the-shelf commercial HMD available (16:1).

A Head-Mounted Display (HMD) can provide a virtual world to the battle manager. AFIT's HMD has already *placed* people into a battlefield. Recorded data from the Air Force Red Flag exercises were used to create the environment. A tracking unit was placed on the helmet to track the user's head position and movement. The user provides commands to the system through the use of a computer mouse. The user can push a mouse button to *jump* into the nearest airplane, or perhaps turn his head to see what is behind him (10:2-3,10-12). This spatial freedom could give the battle manager the ability to *move* through the battlefield and assimilate information effectively.

2.4 Summary

The battle management environment is complex and information filled. Several systems exist to aid the battle manager and staff with the problem using Artificial Intelligence techniques. Effective use of computer graphics also shows promise for helping the battle manager and staff cope with the workload. One effective computer graphics technique discussed here was the concept of a three dimensional virtual environment.

III. Requirements

The BMVS is being designed in an academic environment in order to establish the concept. There is no specific end user and therefore no user requirements exist for the BMVS. The lack of user requirements and the early stage of development allow for a completely wide open design space. While complete design freedom seems at first attractive, chances for success will be greater if the design is constrained somewhat. The design space therefore will be limited to only that which will satisfy the requirements described below.

Because we are generating the requirements as designers rather than having users provide them, the fine line between requirements and design will likely be *stepped over* more than a few times. The requirements will be broken down into two major parts, system requirements and software requirements.

3.1 System Requirements

3.1.1 General. The BMVS should be capable of taking data elements from an Air Tasking Order and displaying a three dimensional, time-dependent representation of the order. Note that this is a representation of the order, to give the battle staff a general idea of what the battlefield *may* look like when the ATO is executed. It can not preview in exact detail how the order will be executed. This is simply not possible with the amount of data available at this planning level. It is intended to provide the battle staff with some insights into the battle situation.

3.1.2 Hardware. Specific selection of computing hardware is deferred until the design. However, the availability of machines limit the selection to one of the following systems:

- Silicon Graphics IRIS 3130 workstation

- Sun 4 workstation with TAAC applications accelerator
- DEC GPX workstation

The system must be selected by how well it meets the following (prioritized with highest first) requirements:

3.1.2.1 The system must be capable of displaying a good quality color graphics image.

3.1.2.2 The system must provide a frame rate high enough (with moderately complex images) to provide acceptable real-time animation. Eight frames a second or higher is desired.

3.1.2.3 The system must support a wide variety of input devices such as keyboard, mouse, Polhemus 3 space tracker, and VPL data glove.

3.2 Software

3.2.1 General. The software for this system is the key element. The hardware is off the shelf and will not likely require any special configurations. The software will be designed with efficiency in mind, because a high frame rate is desirable. The software is not required to be foolproof, that is, extensive error checking is secondary to overall functionality and performance. Specific software requirements are listed below.

3.2.2 Displayable Objects. The BMVS must be capable of displaying objects of four basic classes. First, a static representation of the terrain is required. Second, objects that are movable but are not simulated in a time-dependent manner are required (e.g., SAM sights, targets, etc.). Third, representations of flight paths that the user can modify are required. Finally, objects that are simulated in a time dependent manner, such as aircraft, are required. Table 1 lists the required objects.

Table 1. BMVS displayable objects

Class	Description	Movable	Editable	Time Dependent
1	Terrain	No	No	No
2	Targets	Yes	No	No
2	Threat zones	Yes	No	No
2	Terminal control areas (TCAs)	Yes	No	No
2	Vector airways waypoints	Yes	No	No
2	Low Level Transit Route	Yes	No	No
3	Flight Paths	No	Yes	No
4	Aircraft	No	No	Yes
4	User's point of view	No	No	Yes

3.2.3 Input device support. The BMVS must support a variety of input devices without greatly impacting the main routines. That is, a degree of device independence is required.

3.2.4 Simulation. The BMVS must be able to preview how the execution of an ATO may look. This involves simulation of time-dependent events and near real-time animation.

3.2.5 Parameter files. The ATO information needed for the simulation must be stored in files that the BMVS must be able to access.

3.3 Summary

These requirements will form the basis for evaluation of the BMVS. Chapter V discusses how well the BMVS, as implemented, meets the requirements stated above.

IV. Design

Two basic design approaches were considered for this project. Both approaches are object oriented but, to varying degrees. The first method was a completely object oriented approach to be implemented in the C++ programming language. The second method used some object oriented modules (in C) and reused some procedurally designed software. This section will briefly describe each and explain why the latter was chosen. Then the latter method will be discussed in detail.

4.1 Comparison of Two Object Oriented Techniques

4.1.1 "Full Blown" Approach. The first approach considered was a completely new development using object oriented design. A system that simulates the real world is an ideal candidate for object oriented design. The real world objects are simply modeled as software objects. For example, an abstract data type of an aircraft can be created that has many of the same attributes and functions of a real aircraft (e.g., speed, position, heading, fuel, accelerate, dive). Object oriented design is different enough from conventional design to significantly drive the choice of implementation language.

The C++ programming language has several constructs, absent in C, to support object oriented development. One important construct is the ability to define abstract data types. An abstract data type(ADT) is a set of values and operations applicable to each object (1:28). In standard C the set of values and operations applicable to each object can be easily described in structures. However to describe the operations several *C programmer tricks* must be used and will lower the readability of the product. A second feature C++ provides is class inheritance; objects that are similar can share portions of the abstract data type. For example, airplanes and helicopters might share a *class* called aircraft. The class aircraft describes the common factors such as speed, altitude, etc. Meanwhile the class helicopter contains unique factors such as rotor speed. The C language does not directly support class inheritance. C++ is preferable to C for object oriented designs because of the language support described above.

Although the object oriented design and C++ implementation approach is sound, it is not without problems. Object oriented design provides good control of an object's actions, such as "plane number one please speed up". Some object interactions tend to be awkward, however. A bomb hitting a target cannot directly destroy the target (objects have their own internal set of operations); instead the bomb must request "airfield one, please destroy yourself". Implementing the design in C++ has some problems of its own. C++ compilers are newer and less proven than C compilers. In addition, C++ is not installed on as many computers. It was in fact this second limitation of C++ that drove the selection of method two described below.

4.1.2 Hybrid Approach. The main graphics *engine* in the AFIT graphics laboratory is a Silicon Graphics Iris 3130 workstation. Unfortunately, our attempts to install C++ on the Iris were unsuccessful. That forced the following trade-off:

- Implement with C++ on the other workstations (Sun 4 with TAAC card, or DEC GPX) and suffer performance degradations.
- Or, implement with C on the Iris and modify the design/implementation approach.

Gary Lorimor, as part of his thesis work, compared the performance of the Iris and the GPX workstations. He found that the Iris could produce displays much faster than the GPXs(11:24). During an independent study project, Robert Filer discovered that the Sun 4 with TAAC card performed (in terms of frame rates) 4 to 5 times slower than the Iris(3). Clearly, from the performance standpoint, the Iris was the system of choice.

To achieve the top performance from the Iris, some machine dependent libraries must be used. By using these libraries, a degree of transportabililty is sacrificed. Our decision was that a non-transportable system that works well is superior to a transportable system that performs inadequately. This decision was based on the assumption that no new hardware (money) was available to apply towards the problem.

One side benefit of choosing the Iris was that much of the software used for Gary Lorimor's thesis could be reused. Capt Lorimor's software produced a real-time display of time dependent data. Although his software dealt with replaying recorded events (vs. previewing planned events), several common requirements exist. Both his system and the BMVS must have a method to read in object descriptions from secondary storage. Both must have a way to represent terrain. Finally, both must be able to render and move aircraft images along a time dependent path.

By reusing this software a hybrid design approach emerged. Capt Lorimor's system (hereafter referred to as Red Flag) would be taken as the baseline. As BMVS modules were developed they would be linked into the Red Flag system. This provided a type of rapid prototyping capability in that as the first BMVS module was written the system could be tested as a whole. The Red Flag system is not object oriented so key areas of the Red Flag (such as input devices and aircraft) would be converted to *objects*. Thus we are asking if object oriented design can be retrofitted into existing software. This approach also provides some insight into how practical code reuse is on this scale.

Based on this design approach and the requirement to support multiple input devices, two basic hardware architectures were selected. The first uses the SGI 3130 workstation as the single processing unit (see Figure 2). In this configuration the 3130 handles all the input and output processing. The BMVS is implemented to support the input devices shown (i.e., mouse, keyboard, Spaceball and Polhemus tracker). However, this architecture is flexible enough to allow for the addition of new input devices at a later time. The second architecture uses the SGI 3130 as the main processor, but offloads some of the input handling to a DEC MicroVax (see Figure 3). The MicroVax relays the input device information to the 3130 via an Ethernet connection. The second option should have better performance because the SGI will be freed from the task of dealing with all the input data. The second architecture doesn't greatly impact the software complexity of the BMVS because the Virtual Environment Display System (see Section 4.4) directly supports this architecture.

4.2 Design

Because an existing system is being used as the starting baseline, the top-level design already exists. However, the existing top level design must be analyzed. The results of the analysis help determine if the design is extensible to the BMVS. The results of the analysis show that the Red Flag design can easily support the BMVS. The Red Flag system is a transaction based system. The input devices are polled and then one of several options are executed based on the inputs. This activity of polling and executing is repeated until a quit input is received.

While the top-level design of the Red Flag system works quite well for the BMVS, some of the lower level design does not. In particular the data abstractions used do not correspond to objects (as desired for the BMVS). The following section explains some of the lower-level design issues involved with reusing the Red Flag software.

4.3 Detailed Design

The BMVS system is essentially a virtual world in which various objects reside. The objects are abstract data records, that is collections of attributes to describe the nature and state of an object being modeled. For example, the abstract data record for an aircraft may contain information such as speed, mission, time over target, etc.

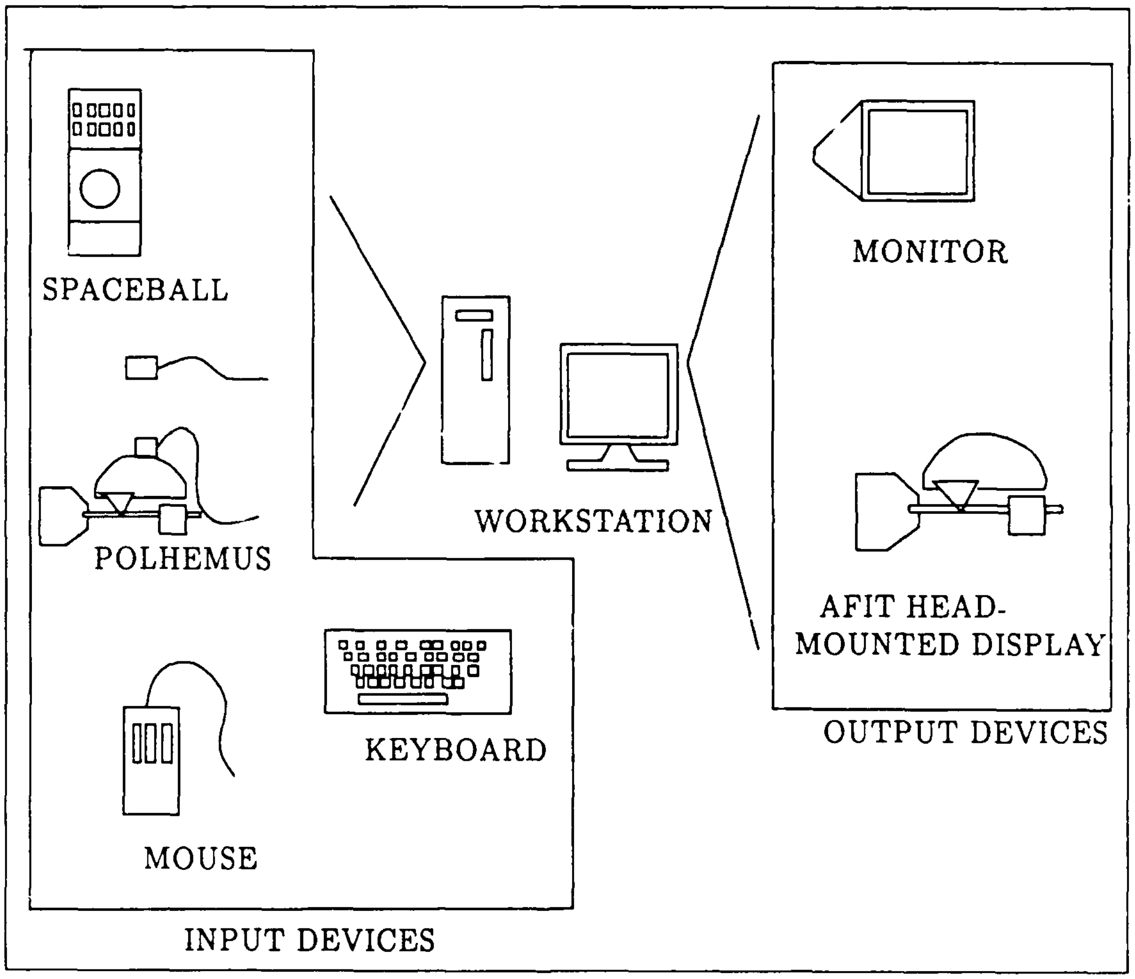

Figure 2. Basic hardware architecture for the BMVS

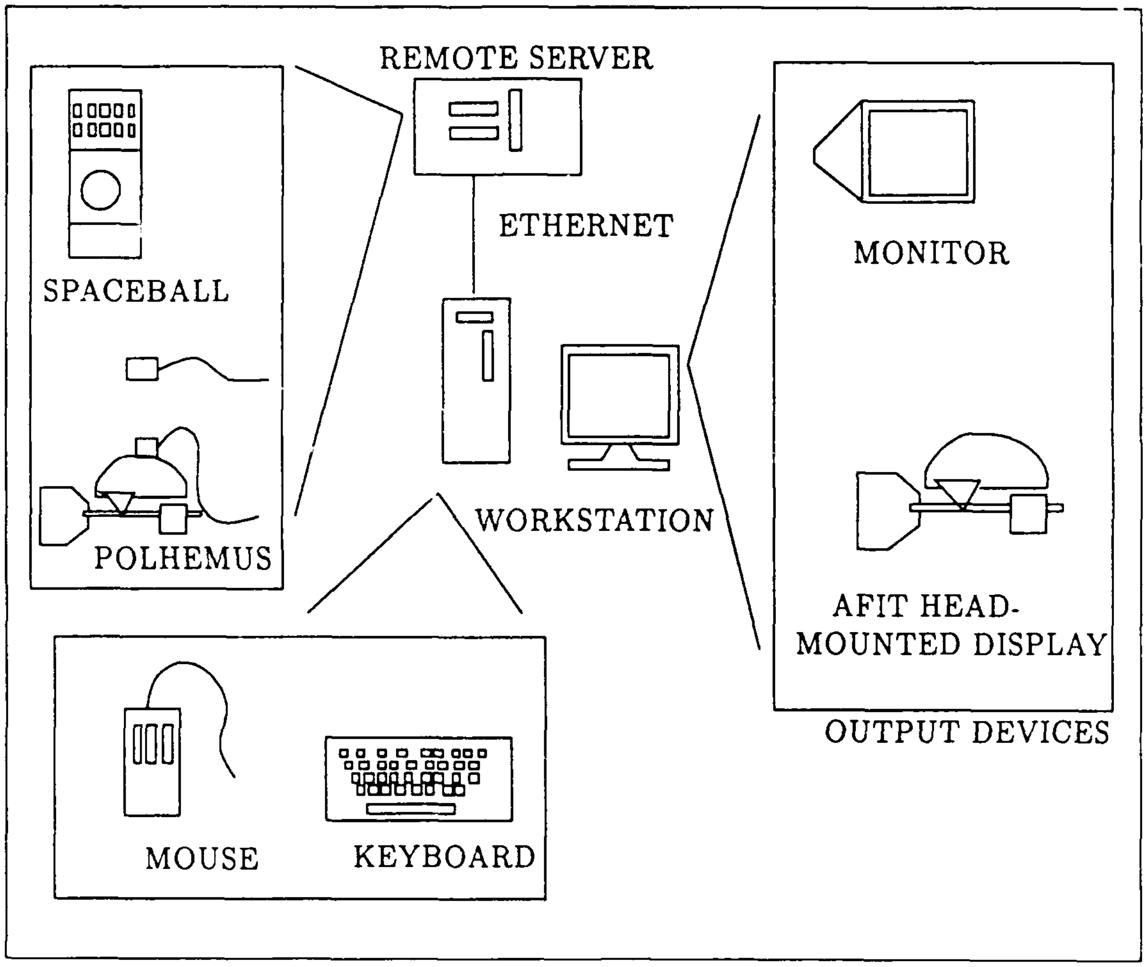

Figure 3. Extended hardware architecture for the BMVS

However, the operations that act upon an object are external to the object's abstract data record. Because the operations are not part of the abstract data record this approach is not totally object oriented.

Because of the importance of the objects in this system, the data structures used to represent the objects will be discussed in detail in the section below. Then implementation issues related to generating flight path data will be discussed. The detailed design discussion will conclude by addressing some issues involved with using the Virtual Environment Display System.

4.3.1 Data Structures. Three important data structures constitute the objects of concern in the BMVS. The object structures consist of the aircraft structure for all the aircraft involved, the path structure for the planned routes the aircraft fly, and the Basic Encylopedia Number (BEN) structure for all the generally static objects such as targets, SAM sites etc. Several other important data structures, which do not constitute objects, will be discussed as well.

4.3.1.1 Aircraft Objects. The aircraft object is essentially an abstract record that contains the information about the aircraft. Some of the information in the record is of use only internally to BMVS (e.g., graphic model number), others are of use to only the user (e.g., mission code) and other elements are useful to both (e.g., aircraft type). See Table 2 for a description of the aircraft record.

Aircraft objects are maintained in a linked list of aircraft as shown in Figure 4. The aircraft are also linked into path objects to access position and orientation information. This external link represents a break in the object abstraction; however, the nature of the problem and memory efficiency concerns warrant the break. Several aircraft may share similar paths; in fact, the aircraft often fly in groups of two or more for protection. By allowing aircraft that fly similar routes to share path descriptions via an external object (rather than replicating path descriptions within each aircraft object) a substantial amount of memory can be saved.

Table 2. Aircraft object data structure

Name	Description
name	aircraft name "F15", "F4", etc.
color	render the aircraft in this color
position	current position of the aircraft (x,y,z)
offset	distance from the path aircraft flies (x,y,z)
model[2]	SGI graphic object number, used to display the aircraft
delta	the distance the aircraft flies in one tick of the clock
remaining_delta	distance remaining to fly in a particular tick
remaining_distance	distance to end of line segment on path
next_plane	pointer to the next aircraft object
pathname	the path this aircraft follows
package	which package this aircraft belongs to
mission	mission identifier
ordnance	what ordnance the aircraft has aboard
gfile	user specified geometry file for the object
speed	aircraft speed (MPH)
index	index into path, how far down the path is the aircraft
selected	boolean flag
time_over_target	planned time over target
start_time	planned start_time
path	pointer to the path this aircraft follows

4.3.1.2 Path Objects. The path object is an abstract record which contains position, orientation, and linkage information about a path. See Table 3 for a description of the path record. The path record contains an array of x,y,z coordinates that specify the positions along the path. Similar arrays are maintained for the set of directional cosines that are used to orient an object placed onto the path. The paths (positions and orientations) are generated from just a few control points or waypoints. The path structure maintains a pointer into the list of waypoints.

4.3.1.3 BEN Objects. An ATO often makes reference to BENs (Basic Encyclopedia Numbers) to identify particular objects (targets, SAM sites, etc.) rather than spelling out the object's details directly in the ATO. Appendix A de-

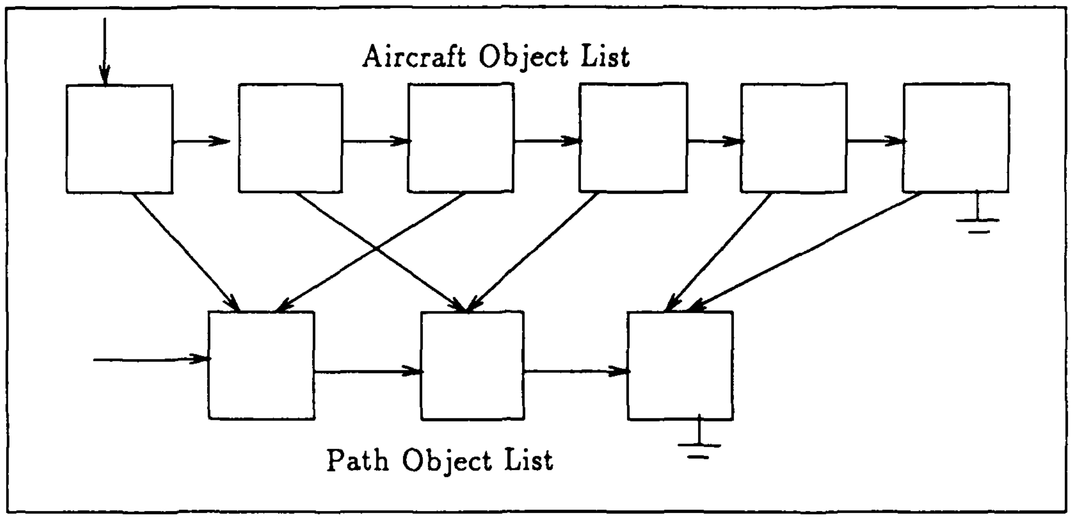

Figure 4. Aircraft list

scribes the typical content of an ATO. The BEN objects in the BMVS are generally simpler than the aircraft and path objects described above. For example, while the user may be able to select and reposition a SAM site, the SAM site is not simulated with the same user independent, dynamic nature that the aircraft is. Also, in most cases the BEN object is a local object as opposed to the path object that may span the theater.

4.3.1.4 Other Structures. Two other structures are very important in the operation of the BMVS and therefore warrant discussion here. The settings structure deals with the user's run-time preferences and the command structure allows different input devices to be used in a consistent manner.

The settings structure is designed as an abstract record that contains all the information pertaining to run-time options. Which input devices are active, what display mode is selected, and how paths should be displayed are examples of questions that are answered with the contents of the setting structure. One benefit

Table 3. Path object data structure

Name	Description
pathname	used to uniquely identify this path
count	number of control points along this path
next_path	pointer to link to the next path in list
checkpoint	pointer to the first control point
object_num	SGI object number, used in displaying the path
dist_to_target	distance along path from start to target
path_length	total length along the path
length_per_segment	distance from one line segment of path to next
active	only active paths get displayed
selected	is this the selected path
color	color to draw the path
point_limit	number of points used to create line segments along path
path_style	which complexity of path to use
cp_saved_color	saves the color of the selected control point
points[MAX_POINTS]	array of x,y,z points to describe the path
delta_direction	direction to fly to get from one point to next
xcos[MAX_POINTS]	x direction cosines (x,y,z) used in orienting aircraft
ycos[MAX_POINTS]	y direction cosines (x,y,z) used in orienting aircraft
zcos[MAX_POINTS]	z direction cosines (x,y,z) used in orienting aircraft

to collecting all the setting information together is that it makes debugging easier. When debugging, a quick check of the settings structure gives a complete picture of how the software is configured.

The command structure provides the interface between the input devices and the BMVS. The protocol is as follows. A device dependent routine is called. If this device has any new input, such as a mouse press on a mouse driver, the input is translated to a command by the driver. Finally, this new command is pushed onto a stack of commands. Once all the input drivers have been polled, the main routine will empty the stack of commands and act on those commands. This cycle of polling all input devices, followed by executing commands continues until a quit command is received. This allows for several (quite different) devices to have the same interface to the main program. It also allows more than one device to send commands to the main program during any given polling cycle.

4.3.2 Generation of Flight Path Data The BMVS and Red Flag systems are similar in several areas. In the area of flight path generation, however, they differ greatly. The Red Flag system has to deal with having too much flight data. The BMVS must, conversely deal with too little data. The Red Flag system has an updated data record available for each high-activity aircraft every one-third second of elapsed time. Two out of three records are thrown out completely to save on storage space. Much of the remaining record is thrown out as well. What is retained is a data set that describes the aircraft's identification, position, and orientation at an instant in time. The BMVS must attempt to give the same feeling of realism with but a handful of data points. An ATO will specify locations of the launching airbase, perhaps a fuel rendezvous, a target location, and a return destination. From these few points the entire path must be generated. In addition, the ATO may only specify a corresponding time for one of the points (the target). Therefore, all other times must be derived from the time over target. Finally, no data is available as to the orientation of the aircraft. The methods of dealing with the position, time, and orientation data are described in detail below.

4.3.2.1 Positions. One way to derive the path positions from just a few points is to linearly interpolate between the points. This is efficient in both time and storage use. However, the results are unsatisfactory (see Figure 5.A). A more common (and acceptable) solution is to use the data points as control points to generate a cubic spline. A cubic spline is a piecewise polynomial that is useful in computer graphics due to its continuity characteristics. Where one piece of the cubic spline meets the next piece there is no discontinuity between the functions or between the first derivitives of the functions (some splines are continous in the second derivitive as well). These continuity characteristics make the resulting curve

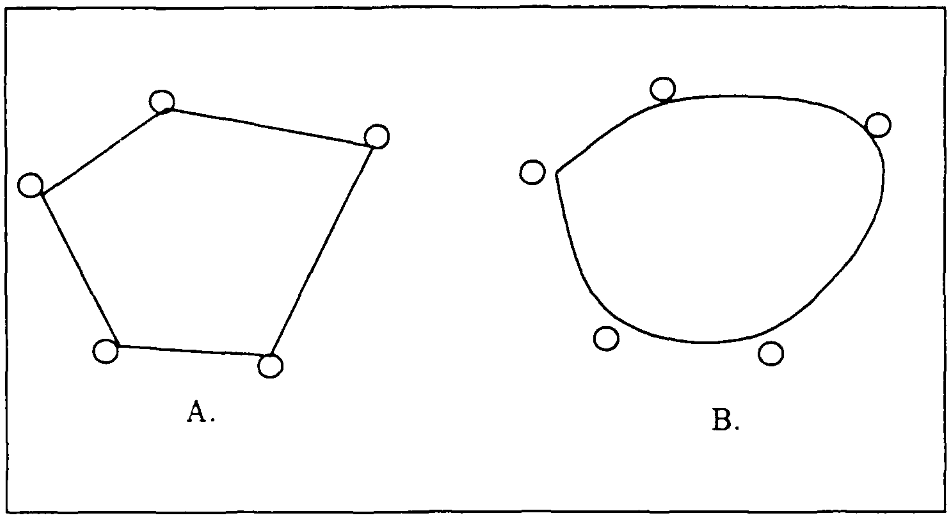

Figure 5. Straight vs. curved line approximations to the path

look smooth and natural. The curved path shown in Figure 5.B looks much more natural for an aircraft path than the straight line approach. Note that the curved path only *looks* better; it doesn't convey more accurate information than the straight path since no real information exists between the points. Because the curved path is used only for aesthetic purposes, the selection of which cubic spline to use is not critical. For convenience we tailored an existing program (13) to meet the needs of the BMVS. The program can generate Cardinal splines or Beta splines, either of which appears satisfactory.

4.3.2.2 Time Data. Generally, the only time included for an aircraft in an ATO is the Time Over Target (TOT). Aircraft on alert are the exception (since they have no predetermined target) and usually have specified start time or window of vulnerability. In either case only one time is given from which all other times must be derived.

The approach taken in the BMVS to calculate the times is straight forward. First the aircraft is checked to see if it has a start time, a time over target, or both.

If it has only a start time, the times along the path will be determined based on the distance from the starting point and the speed of the aircraft. If only a TOT is specified, the distance from start to target along with the speed is used to determine the start time. Once the start time has been determined, the rest of the times are found as in the first case. If both the start time and TOT are specified, the speed flown is calculated from the time difference and the distance flown from start to target. It is important to note that the start time and TOT are inputs from the user and therefore are given more weight than the distance of the path since it was artificially generated. It is possible for the BMVS to calculate an unrealistic speed (e.g., 55 knots for an F15). This is absolutely acceptable (although not realistic) to show the progress of the aircraft toward the target. Obviously the real aircraft would have to fly a much longer path at a higher speed to meet the two time requirements. However, at the times of high interest (i.e., start time and time over target) the simulated aircraft would have an acceptably accurate position.

4.3.2.3 Orientation Data For a realistic look, it is not enough to simply curve the path of the airplane, the airplane must be oriented correctly as well. It would look very unnatural for an airplane to go through a tight turn without banking. The bank angle an actual aircraft achieves through a turn is dependent on several factors. The radius of the turn, aircraft design, speed of the aircraft, and amount of applied rudder are but a few of the factors related to bank angle. For a planning tool (vs. a flight simulator) the dynamics of the aircraft do not need to be accurately modeled. The requirement therefore is to produce something more believable than an aircraft making level turns without the necessity of dynamic modeling.

The middle ground used for this project is to use only path data to determine the bank, ignoring all aircraft data. The method to determine the bank angle as well as other orientation data (i.e., heading and angle of attack) is as follows. First the aircraft is assumed to have an initial orientation of straight and level with an arbitrary heading(Figure 6A.). This is represented by a vector in the Z direction (the tail vector $\vec{T}$). Then the current point and the next two points along the curved path are examined. From these three points a velocity vector ($\vec{V}$)and acceleration vector (A) are calculated(Figure 6B.). The cross product of the tail and velocity vector results in a wing vector($\vec{W}$) (Figure 6C.). The velocity vector and the wing vector are then crossed to calculate a new tail vector which insures a mutually perpendicular set of basis vectors (tail, velocity, wing). These vectors are used to orient the aircraft with the fuselage along the $\vec{V}$ vector, the tail along the $\vec{T}$ vector and the wing along the $\vec{W}$ vector (Figure 6D.). To determine the bank angle, the magnitude of the acceleration vector is examined. The aircraft basis vectors are then rotated about the velocity axis based on the magnitude of the acceleration vector (Figure 6E.). The function that maps acceleration magnitude to bank angle was experimentally derived to produce acceptable results.

4.4 The Virtual Environment Display System

The Virtual Environment Display System (VEDS) is a collection of software libraries and hardware devices being developed to support virtual environment systems such as the BMVS (4). Because VEDS is being developed in parallel with the BMVS, the risks and benefits of using the VEDS for the BMVS had to be analyzed. A conservative approach was taken. Routines that could benefit most from the use of VEDS were scheduled later in the development. For example, the keyboard and mouse routines do not require VEDS support (they are supported by the Silicon Graphics Library) and were developed first. The *hooks* for other (VEDS supported) devices were put in place, but the actual development was delayed until the VEDS drivers matured. Figure 7 shows the relationship of the BMVS, the VEDS, the SGI libraries, and the operating system. Even with this conservative approach, one key dependency (and therefore risk) on the VEDS was accepted. The benefits of the VEDS Polhemus driver outweighed the risks.

To help illustrate the importance of the VEDS Polhemus driver, some back-

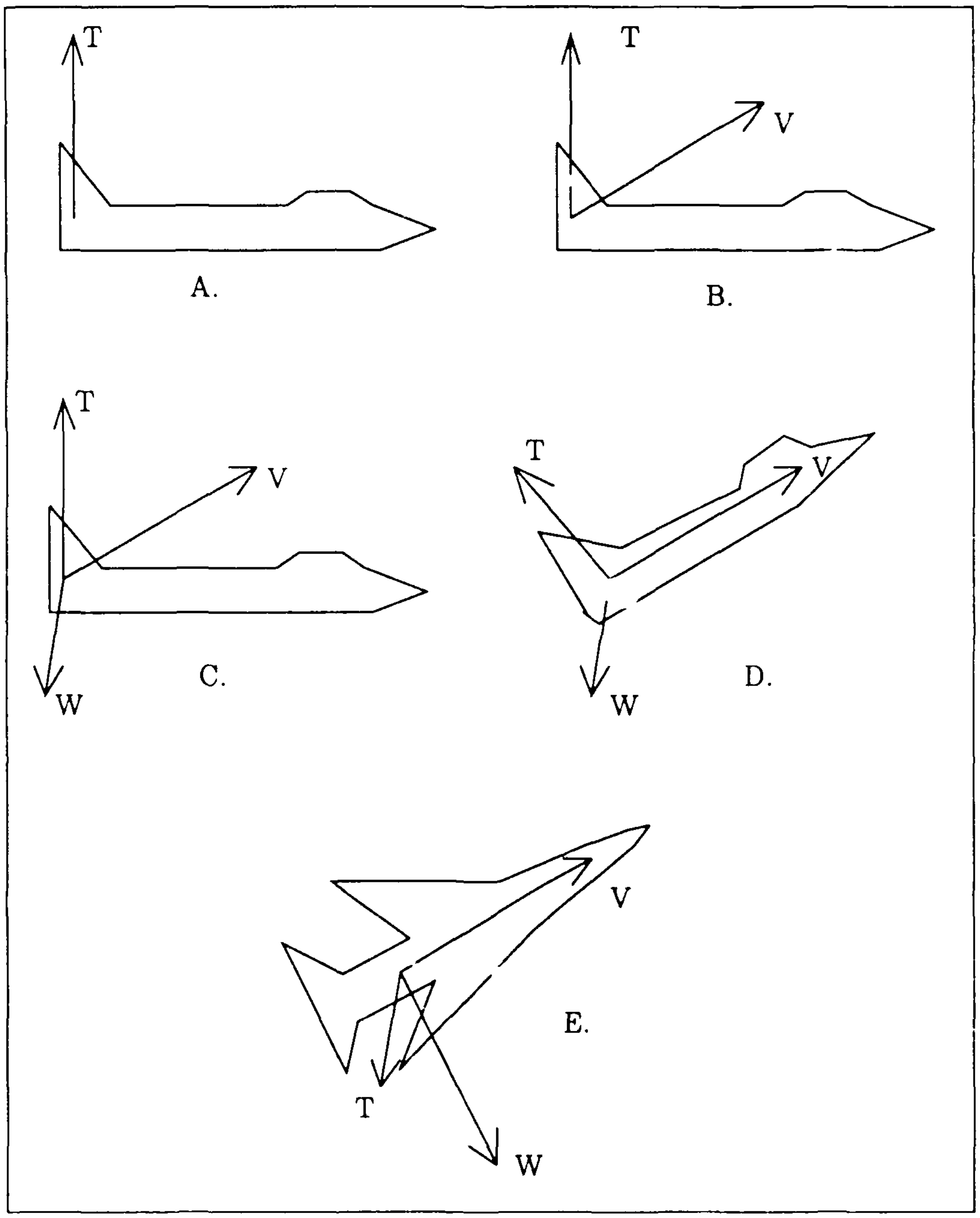

Figure 6. Steps used to orient the aircraft

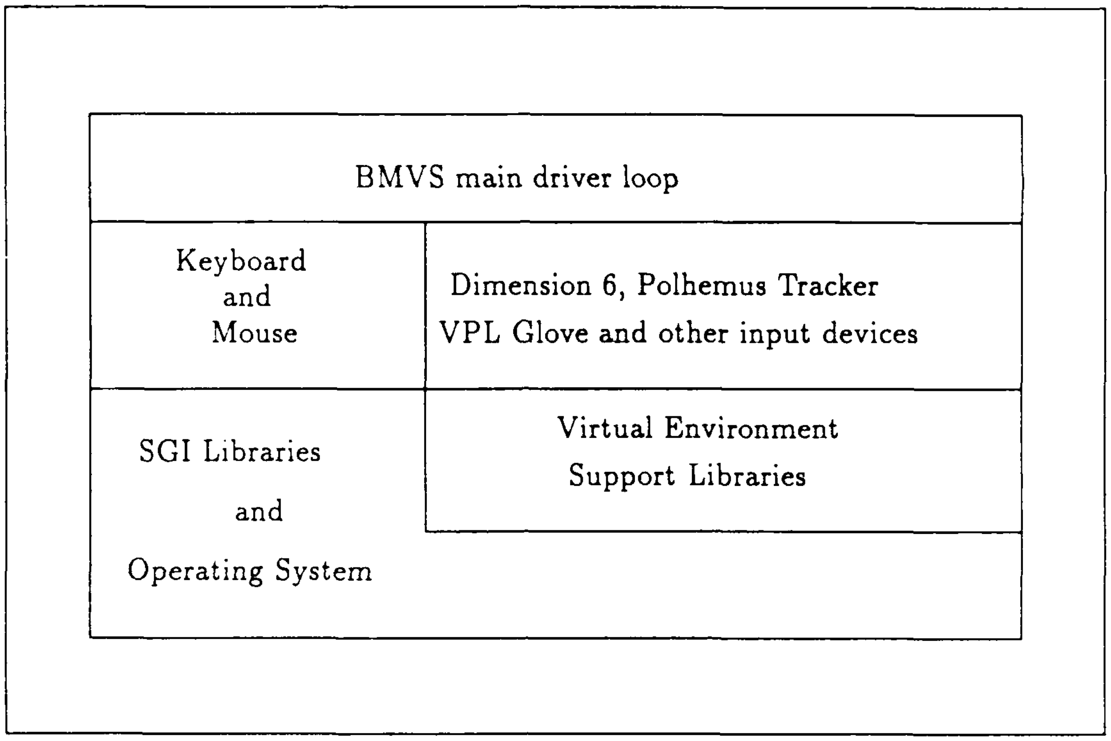
BMVS main driver loop
Keyboard and Mouse
Dimension 6, Polhemus Tracker VPL Glove and other input devices
SGI Libraries and Operating System
Virtual Environment Support Libraries

Figure 7. BMVS, library interfaces

ground on the Polhemus tracker is needed. The Polhemus tracker uses a low frequency magnetic field to track the position and orientation of a sensor. The Polhemus operates in either a polled or a continuous mode. Both modes introduce some problems for the BMVS. When in continuous mode the Polhemus will output position and orientation data whether the BMVS wants it or not. The BMVS is then responsible for sorting out the data, queuing it up, and flushing all but the latest. When operated in polled mode the Polhemus returns data only when requested, but only after a significant delay. The Red Flag system uses the polled method and circumvents the delay problem by doing some useful work between the polling and the reading operations. This solution works well but is not without problems of its own. How much work should be done between polling and reading? What happens when other input devices are used in conjunction with the Polhemus? What happens when the

Polhemus response rate is cut in half (necessary when using a second sensor)? These are just a few of the questions that must be answered if the Red Flag approach were used in the BMVS. Fortunately, the VEDS library offers the best solution.

The VEDS library provides a Polhemus driver routine that abstracts all the unecessary details of communicating with the Polhemus away from the application builder. All the application program does is call a VEDS library routine. The VEDS library routine handles the timing details involved in communicating with the Polhemus tracker. The VEDS Polhemus driver can even reside on a separate computer so as not to compete with the BMVS for the CPU.

V. Results

This chapter addresses how well BMVS meets the requirements stated in Chapter 3. Additionally, the effectiveness of the code reuse described in Chapter 4 will be discussed.

5.1 Image Quality

Section 3.1.2.1 states a subjective requirement for a *good* quality graphics image. Figure 8 is a photograph taken directly from the BMVS screen. This photograph, along with demonstrations to various audiences, indicates that the image is of sufficient quality to allow object recognition.

5.2 Input Devices

Section 3.1.2.3 requires the software and the system to support multiple input devices. As implemented, the BMVS supports mouse, keyboard, Dimension6 and Polhemus input devices. Section 5.4 discusses effects of the input devices on performance.

5.3 Displayed Objects

Section 3.2.2 lists the requirements for all displayed objects. Geometric models for all of the required objects have been implemented with the BMVS. Some of the objects have multiple representations, allowing the user to choose the best representation for his needs. For example, there are four possible representations for path objects.

5.4 Frame Rate

The frame rate requirement stated in Section 3.1.2.2 is a minimum of eight frames a second. Table 4 lists frame rates observed under various conditions. Several

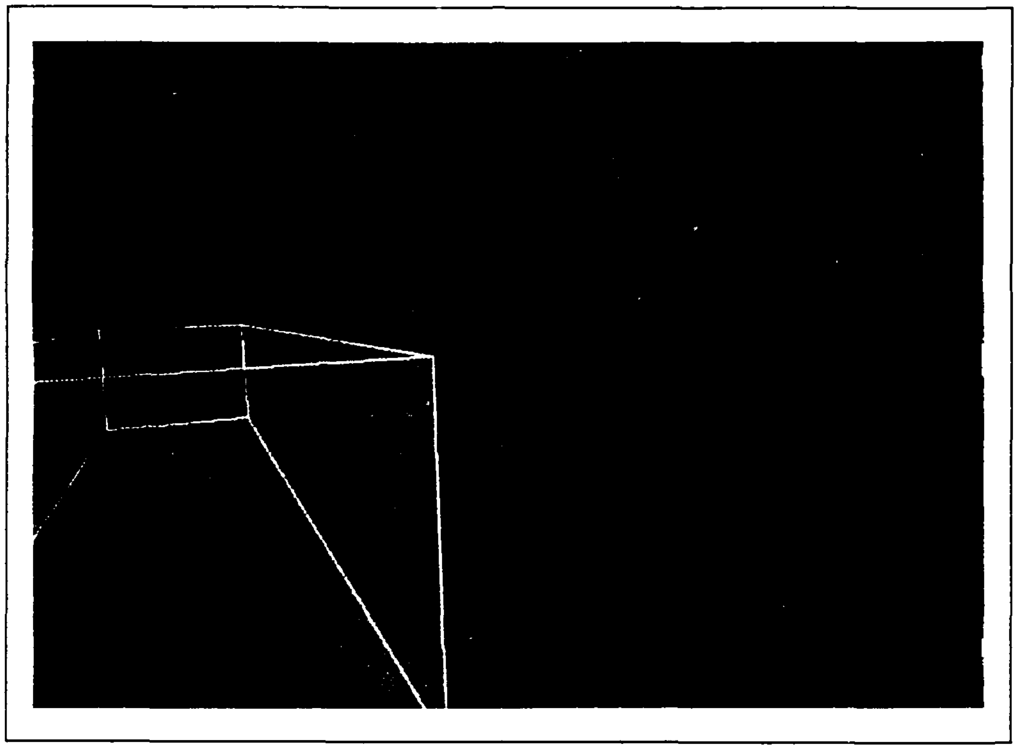

Figure 8. Sample BMVS screen

factors affect the frame rate. The complexity of the graphics image is the most critical factor. The complexity of case one was relatively low (approximately 200 filled polygons and 500 vectors). Case 2 was similar in all respects except that approximately 3200 additional vectors were introduced to represent the paths.

The BMVS meets the 8 FPS requirements in most cases. The cases where it doesn't meet the requirement are due to a high degree of scene complexity. Therefore in certain situations the user may have to decide between increasing the scene complexity (at the cost of frame rate) or keeping the frame rate high (at the expense of some scene complexity).

Table 4. Frame rates under various conditions

Case	Conditions	Time	Frames	Frames/Sec.
1	2 paths (not shown), 3 aircraft active, Green floor shown, 2 TCAs, 4 SAMs, 2 targets, and one LLAR in *world* mouse and keyboard inputs only see Figure 9	40	600	15
2	same as case 1 but paths shown see Figure 10	91	600	6.5
3	same as case 2 with autofly active	93	600	6.5
4	same as case 1 with autofly active	37	600	16.2
5	same as case 1 but Dim6 (louvre as server) as additional input	52	600	11.5
6	same as case 1 but Polhemus (using louvre server)	40	600	15
7	same as case 4 but Polhemus (using louvre server)	45	600	13.3

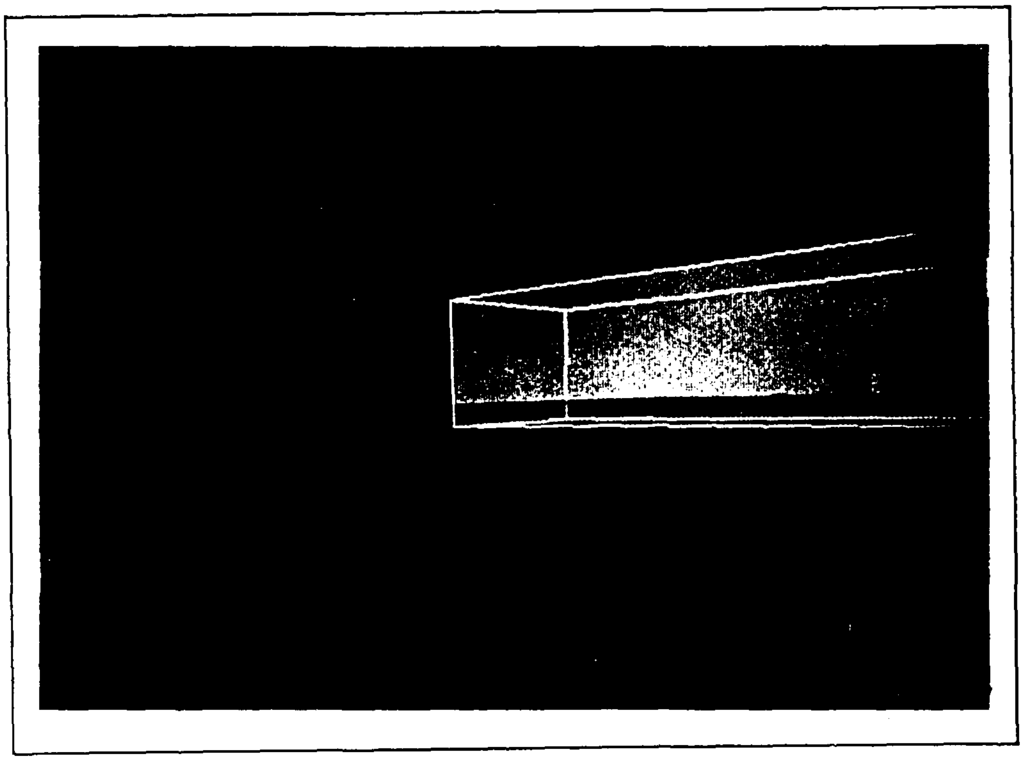

Figure 9. Moderately complex scene no paths shown

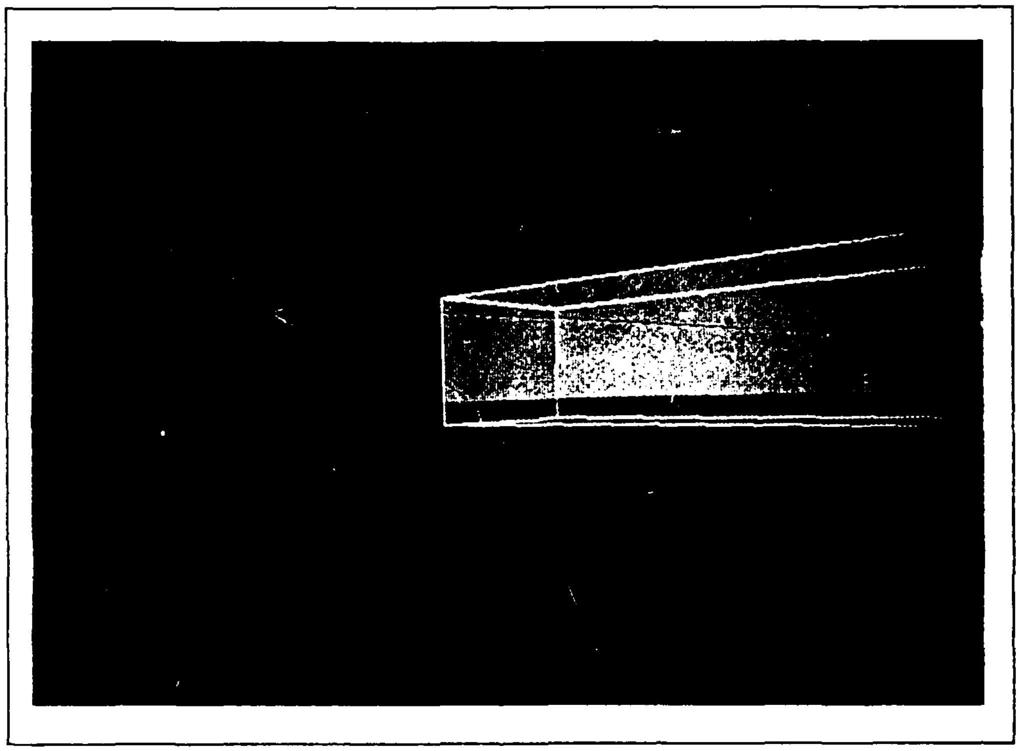

Figure 10. Moderately complex scene but paths shown

5.5 Code Reuse

One benefit of the hybrid design approach taken for the BMVS, discussed in Section 4.1.2, was the opportunity to reuse existing software. This section will discuss the amount of code reuse as well as any impacts, both positive and negative.

Table 5 lists the BMVS modules in three categories: totally new modules, the reused and modified, and the reused unmodified. In general, the table is ordered left to right in time and effort required (i.e., it takes longer to write new code than it does to use existing code unmodified). Some of the existing modules required so much modification that the final version resembled the original version in name only. The two most notable modules of this type are the driver.c and the makeTransform.c modules. These modules are counted as reused even though the effort was as much or more than that needed for a new module. This effect is balanced out by the fact that some modified code had only one or two lines that had to be modified (e.g., make_filled.c).

Of the 5100 source lines of code (SLOC) for the BMVS, over 2200 were created with some degree of code reuse. The percentage of code reuse then is over 40%. Eight percent of the total code was completed by reusing code with no modification. If driver.c and makeTransform.c were counted as new code, the resulting percentage of code reuse is still over 25%. These statistics indicate that reuse of C code on a small project such as this can be practical.

Table 5. New code development vs. reused

New		Modified		Unmodified	
Name	SLOC	Name	SLOC	Name	SLOC
apply_settings.c	22	do_undo.c	112	data.c	122
ben.c	94	graphics.c	93	lookat.c	182
dim6.c	186	makeTransform.c	277	make_wire.c	57
distance.c	23	makebox.c	126	writestring.c	37
do_polhemus.c	57	make_filled.c	54	swept.h	48
fullmouse.c	217	driver.c	612	vector.h	5
glove.c	9	make_objs.c	165		
init.c	114	set_plane.c	77		
joystick.c	9	spline.c	133		
keyboard.c	454	externs.h	73		
make airzone.c	177	graphics.h	108		
make_mover.c	57				
make_pathstyle3.c	66				
make_time.c	92				
modify_settings.c	83				
makepath.c	271				
newpath.c	69				
readato.c	232				
savefile.c	178				
set_clock.c	77				
support.c	100				
set_auto.c	53				
constants.h	20				
protos.h	21				
structures.h	187				
macros.h	7				
total:	**2875**	**total:**	**1828**	**total:**	**451**

VI. Conclusions and Recommendations

6.1 Conclusions

Overall, this thesis project achieved the desired objectives. The basic questions (see Section 1.3) have been answered. Yes, a software system can be designed to provide a battle manager some form of preview to an ATO execution. Yes, the *virtual world* interface provides for a natural man/machine interface. Yes, the same concepts used to generate a *virtual military environment* can be used to create a non-combat *virtual environment.*

6.1.1 ATO Preview The BMVS is not the magical *crystal ball* mentioned in the beginning of this thesis. For example, the user is given no predictive view of how the enemy may react during ATO execution. However, the BMVS provides a preview of where key resources will be at key times. An error or omission in an ATO that has a single aircraft arriving at the target without support aircraft could be easily spotted. A conflict in the use of airspace is another problem that could be easily spotted.

6.1.2 Virtual World Interface The virtual world interface in the BMVS is its strongest feature. The user is *placed* into an environment in which he can move about and interact with objects. He can *travel miles* in a few seconds or *move* a SAM site at the push of a button. In short, the user is provided with a powerful interface in which to view and manipulate the environment. Appendix B, the BMVS Users Manual, contains a complete description of the possible interactions.

6.1.3 Civilian Applications Several features were built into the BMVS to allow for support for non-combat applications. First, non military BEN objects and their corresponding geometry descriptions were included. This included Terminal Control Areas (TCAs) and vector airway waypoints. Second, a path representation was developed that resembles the representation of vector airways in aeronautical charts. Finally, the BMVS allows the user to specify the geometric description for a given object. This would allow the user to add new objects of interest into the environment such as obstructing towers, civilian aircraft geometry, etc.

6.2 Recommendations

The BMVS system shows that there is promise in the area of ATO previewing. However, there are several areas that warrant further investigation. A few possible areas to consider are as follows:

- While the user interface for the BMVS is more flexible than that of its predecessor the Red Flag system, there is still room for improvement. For a system like the BMVS, providing the user with an interface he is comfortable with is very important to his acceptance of the overall system. Consider implementing additional input devices into the BMVS to allow more choices and hence increase the likelihood of meeting the user's expectations. The BMVS system already has *hooks* in it for the VPL Data Glove and a joystick. The method of implementing new devices is described in Appendix B, The BMVS Users Manual.
- Additional benefits may be gained by combining some of the replay capabilities of the RED FLAG system with the preview capabilities of the BMVS. By comparing the preview of the ATO execution with an actual ATO execution, the concept could be further validated and refined.
- Transfer the software to a machine with greater graphics capability. Occasionally the lack of hidden surface removal in the BMVS is distracting. State of the art graphics workstations can produce real-time images (of the complexity found in the BMVS) with the hidden surfaces removed.

Appendix A. *Typical ATO Format*

The information below is representative of the information contained in an ATO. Actual formats vary from theater to theater but the information is essentially the same. To run BMVS information would be extracted from the ATO and put into the necessary files.

A.1 SPINS – Special Instructions

SPINS contain essential information which:

- apply to all flights
- apply to a group of flights performing similar tasks
- apply to a single flight out too lengthy to put in mission format

A.2 Mission Line

Example	MSN	1001	8	HUR	2 F-4D	Bravo 5	PKG OZ	INTSTK	F	15 Dec
Field	A	B	C	D	E	F	G	H	I	J

The field descriptions are as follows:

A. Line name (what information follows)

B. 4 digit mission number, used to reference missions in ATO

C. Unit identifier (normally a wing's numerics)

D. 3 letter location identifier for the base where the mission will originate

E. Lists the number and type of aircraft

F. Call sign

G. Package ID

H. Mission type (see below)

I. Service of the unit (F=USAF, M=USMC, N=USN, A=ARMY, X=other)

J. Date of execution (blank if flown daily)

A.3 Ordnance line

Example	ORD	SCL 555	555/509
Field	A	B	C

The field descriptions are as follows:

A. Indicates ordnance line

B. Standard conventional load, written out ordnance or an ordnance code specified in SPINS

C. SCL for 1st and 2nd aircraft in flight

A.4 Time Over Target Line

Example	TOT	0800-0930	AB 15,16	BEN 0380-00000	SAM	BT 15
Field	A	B	C	D	E	F

A. IDs Time Over Target line (TOTs are local time)

B. Time over target (1 time indicates a hard time, 2 times denote a block time

C. Search area for armed recce missions

D. Basic encyclopedia number (from tactical target list)

E. Target description may be included

F. Box location of "BEN" target

A.5 Alternate Time Over Target Line

Example	TOT	0800	cs609907	TANKS	IUF0234PSAM	CFA
Field	A	B	C	D	E	F

A. IDs Time Over Target line (TOTs are local time)

B. Time to plan release of ordnance

C. Location of target

D. Target description or abbreviation

E. Army request number used to ID air support requests

F. Field army to which the mission is distributed

A.6 Airborne Alert Line

Example	ABA	0800-0900	orbit pt 3	Airborne Alert
Field	A	B	C	

A. IDs line

B. Vulnerability time to be in the assigned area

C. Assigned orbit points

A.7 Quick Reaction Posture Line

Example	QRP	1800-2400	3	Alert Status E4D/E3D/B2D/B1D
Field	A	B	C	D

A. IDs line

B. Indicates beginning and end of QRP liability period

C. Expected number of missions during the period

D. Alert status codes

1st char	2nd Char	3rd char
A - 3min	DEFCON LEVEL	D - Day
B - 5min		N - Night
C - 8min		C - Continuous
D - 10min		
E - 15min		
F - 30min		
G - 1hr		
H - 3hr		

A.8 Forward Air Controller Line

Example	FAC	3000	Bronco 31	Orbit Pt 2	TAB 207/209
Field	A	B	C	D	E

A. IDs line

B. The support FAC(AFAC) mission number, reference the CASFAC tasking in the ATO

C. AFAC callsign

D. Is the CAS orbit point to which the fighters are to report.

E. UHG TADS (primary and secondary) to contact FAC

A.9 IFF/SIF/LINE

Example	MOD2	0410	0411
Field	A	B	C

A. IFF line identifier

B. Mod2 squawk for 1st aircraft

C. Mod2 squawk for 2nd aircraft

Appendix B. *BMVS User's Manual*

The BMVS is a very flexible software system. The more flexible a software system becomes, however, the more burden is put on the user to cope with the complexity of the system. For example, no one would accept a word processing system that only allowed the user to save files to the hard disk. By adding the flexibility to save to any disk drive, the user now has to learn the method of communicating the desired drive to the program. Actually word processors are typically very flexible and offer more features than the average user will ever use or need to know how to use. So it is with the BMVS, that is the user can run BMVS with a knowledge of a subset of the complete capability. In that spirit this manual is organized into three categories based on user expertise: Beginning (just what it takes to get BMVS running), Intermediate (for users that may want to run BMVS under various configurations), Expert (for experienced users who may even want to add a new input device). The manual concludes with an explanation of the file formats.

B.1 Beginning Users

This section is meant to get the user *up and running* quickly. Therefore the reader of this section will not have all the *power* of the BMVS at his or her disposal. This section is broken down into what is needed to run and how to run BMVS.

B.1.1 What Is Needed to Run BMVS. The minimum hardware configuration is a SGI 3130 workstation. The BMVS will look for several files in the current directory, some of which must be present. (Table 6 lists the files the BMVS will search for).

B.1.2 How to Run BMVS. The simplest way to run BMVS is to simply type "bmvs" with no options. The program will run using the default options (as modified by paramfile, if present). Running under the default options, the keyboard and mouse are the active input devices and the workstation screen is the output device.

Table 6. Required Files

Filename	Purpose	Required
paramfile	Sets software configuration	NO*
benfile	defines locations of SAM sites and similar objects	NO*†
aircraftfile	Defines the aircraft objects	YES*†
pathfile	Defines the paths the aircraft follow	YES*†
f15	Geometry file	YES
f16	Geometry file	YES
f5	Geometry file	YES
f4	Geometry file	YES
a10	Geometry file	YES
f111	Geometry file	YES
waypoint	Geometry file	YES
target	Geometry file	YES
sam_site	Geometry file	YES
tca	Geometry file	YES
RangeData	Geometry file	YES
bigbox	Geometry file	YES
* filename may be changed by command line options		
†filename may be changed by paramfile contents		

B.1.3 Mouse Control. This section describes the basic mouse button *bindings* (i.e., which mouse button does what) for the BMVS. While these bindings are set up to be easily modified, for the beginning user we will treat the bindings as absolute. Table 7 lists all the standard mouse button bindings. Note that only the mouse buttons are used; no mouse rolling is used. As Table 7 indicates, if the user presses the right mouse button, the scene will pan to the right. Similarly, if the user presses all three buttons at once a pop-up menu appears. The operation of the pop up menu is discussed below. Table 7 also mentions *selecting* a path object. The concept of selected objects is discussed below as well.

B.1.4 Menu Operation. Once a pop-up menu appears, the mouse button bindings change temporarily to allow the user to select the desired menu item.

Table 7. Mouse bindings

Buttons Pressed	Action
None	No action
Right	Pan to the right
Left	Pan to the left
Center	Move forward through scene
Left and Right	Move backwards through scene
Right and Center	Pan down
Left and Center	Pan up
Left, Center, and Right	Access menu

The right button is pressed to cycle through the various selections. Once the desired item is highlighted, the user presses the left mouse button to select that item. The menu then disappears and the mouse returns to normal operation. See Figure 11 for an example of the pop-up menu. See Table 8 for an explanation of each menu item.

B.1.5 Selected Objects. At times the user may want to retrieve information or edit a given path, aircraft, or BEN object. To do this he must somehow indicate to the program which path, aircraft, or BEN object is of interest. This is handled in the BMVS with the concept of the *selected object.* Only one path may be selected at any given time. Likewise only one aircraft and one BEN object may be selected. Each class of objects have a corresponding select command. The select command de-selects the currently selected objects and selects the next object in the list. The program *cycles* through the objects as the user presses the select command such that all of the objects are available for selection. Note that the selected objects are identified by their yellow color yellow.

B.2 Intermediate Users

This section is designed for those familiar with the basic operation of the BMVS but want to use some of the more advanced features. Keyboard, Spaceball,

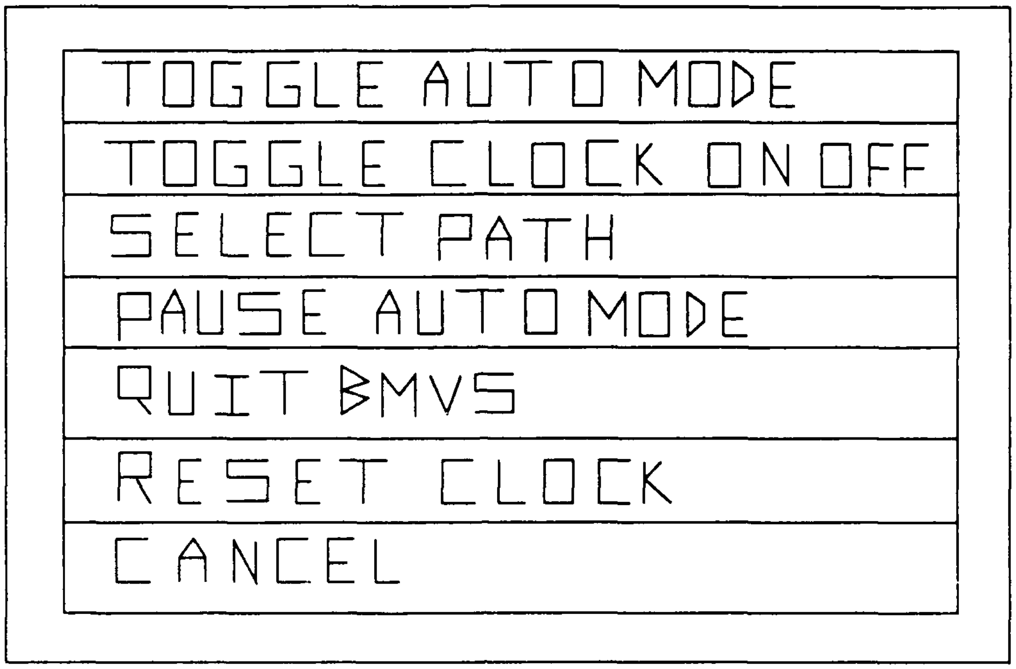

Figure 11. Example menu

Table 8. Menu actions

Menu Item	Action
Toggle auto mode	Turns on or off the autofly feature (where the viewpoint follows a path at a scale speed)
Toggle clock	Turns the clock on and off; when on the active aircraft fly along the paths; when off the aircraft are *frozen*
Select Path	Each time this item is selected a new path is selected. The selected path is the path followed during autofly mode
Pause auto mode	Temporarily holds viewpoint in position when in autofly mode
Quit BMVS	Exit the program
Reset Clock	Resets the clock to allow *instant replays*
Cancel	Exit the menu with no other action taken

and Polhemus input device operation will be discussed. In addition some of the run-time configuration options will be explained.

B.2.1 Keyboard Operation. The keyboard is a key input device for the BMVS as it is the only device where the user can access all of the features. Table 9 lists all the standard key bindings. All commands are initiated with a single keystroke. Some selected commands, set time for example, prompt the user for additional information. These *extended commands* are explained in detail below.

B.2.1.1 Extended Commands. A few of the BMVS commands require more information that can be provided with a single keystroke. These are called extended commands, and require the user to respond to a prompt. Note that these commands use a very unforgiving interface (not even backspace is supported for example). However, the commands are short, so if an error is typed hitting return and re-initiating the command is not difficult. The extended commands are as follows:

c: **set clock** The system will prompt for a time in the format HH:MM:SS the user may;

hit return Resets clock to 00:00:00 (same as R reset clock command)

enter hours HH only Time becomes HH:00:00

enter hours and minutes HH:MM Time becomes HH:MM:00

enter hours, minutes, and seconds HH:MM:SS Sets time to HH:MM:SS

e: **edit run-time configuration** The user will be asked to enter a keyword followed by one or more values. The keyword/value format is the same as the paramfile format (table 14) discussed in the file format section. However not all of the keywords apply. For example the AIRCRAFTFILE: keyword is not available as the aircraft file is read only during program start-up. The applicable keyword/value combinations are listed in table 10.

Table 9. Keyboard functions

Key	Action
a	toggle auto mode
b	move backward through scene
B	select the next BEN object
c	set the clock *
d	pan the viewpoint down
e	edit run-time configuration *
f	move forward through scene
F	move forward through scene Fast
i	get info on selected BEN object
I	get Info on selected plane
k	move selected control point in pos. x direction
K	move selected control point in neg. x direction
l	pan the viewpoint to the left
m	move selected control point in pos. z direction
M	move selected control point in neg. z direction
n	move selected control point in pos. y direction
N	move selected control point in neg. y direction
p	pause auto mode
P	remake selected Path
q	quit BMVS
r	pan viewpoint to the right
s	select next path
S	Select next plane
t	start/stop time
u	pan viewpoint up
w	write out current state of BEN objects *
W	Write out current state of path objects *
x	moves selected BEN object in positive x direction
X	moves selected BEN object in negative X direction
y	moves selected BEN object in positive y direction
Y	moves selected BEN object in negative Y direction
z	moves selected BEN object in positive z direction
Z	moves selected BEN object in negative Z direction
* indicates an extended command	

Table 10. Runtime modifications

Keyword	Values	Description
MODE:	String	"NTSC" for NTSC mode
BLUE_BACK:	Boolean	0 for black background, 1 for blue
CHECKPOINT_ACTIVE:	Boolean	Spline control points shown or not
PATHS_ACTIVE:	Boolean	0 for no paths, 1 shows paths
BEN_ACTIVE:	Boolean	1 to show BEN objects
SAM_ACTIVE:	Boolean	Reserved for later use
AIRPORTS_ACTIVE:	Boolean	Reserved for later use
AIRCRAFT_ACTIVE:	Boolean	Reserved for later use
TICK:	Integer	Tenths of seconds per frame (default 10)
PATH_STYLE:	Integer	Which path style to use (default 1)

w: write BEN file The user enters the desired filename and hits return.

W: write path file The user enters the desired filename and hits return.

B.2.2 Dimension6 input device. The Dimension6 (also referred to as the Spaceball) input device provides six degrees of freedom (3 translation, 3 rotation). It reports a proportional value in each degree of freedom based on the force applied. It has eight pushbutton inputs as well. Table 11 describes the functions of the Ball and the buttons.

B.2.3 Polhemus input device. The Polhemus is the simplest input device from the user's perspective. The user turns his head a given direction and the viewpoint is altered to match that turn. No other BMVS features are accessed with the Polhemus.

B.2.4 Run-time options. The BMVS will, by default, come up supporting the mouse and the keyboard only. If the user wants to activate the Spaceball or Polhemus (or deactivate the mouse or keyboard) the BMVS must be told via the paramfile. The method for activating the Spaceball and Polhemus are similar so a detailed discussion will be given for the Spaceball only. To activate the Spaceball

Table 11. Spaceball functions

Input	Action
Slide ball forward	Move forward through scene
Slide ball backward	Move backward through scene
Rotate ball forward	Pan viewpoint down
Rotate ball backward	Pan viewpoint up
Rotate ball left	Pan viewpoint left
Rotate ball right	Pan viewpoint right
button 2	Toggle autofly mode
button 3	Start/stop clock
button 4	Pause autofly mode
button 5	Select next path
button 6	Mode change – changes ball bindings to those shown in table 12
button 8	Quit BMVS

Table 12. Spaceball functions (alternate mode)

Input	Action
Slide ball forward	Selected BEN object moves in positive y direction
Slide ball backward	Selected BEN object moves in negative y direction
Slide ball right	Selected BEN object moves in positive x direction
Slide ball left	Selected BEN object moves in negative x direction
buttons	Have same function as in table 11

the USE_DIM6: keyword in the paramfile is followed by a value of 1. This tells the BMVS to use input from the Spaceball in conjunction with the other active input devices. Additionally the BMVS must know to which port the Spaceball is connected. The keyword DIM6_PORT: is followed with a string which equals either the tty port the Spaceball is connected to or the word "REMOTE" to indicate the data will come from a remote server. Remote operation is preferred when possible for best overall performance of the BMVS. Examples of both local and remote use of the Spaceball is given below.

B.2.4.1 Local Mode. All that is needed to initiate local mode is to set the proper paramfile keywords as shown in this example:

```
USE_DIM6: 1
DIM6_PORT /dev/tty02
```

B.2.4.2 Remote Mode. For remote mode the paramfile keywords as set as shown below:

```
USE_DIM6: 1
DIM6_PORT REMOTE
```

In addition the remote server must be started as well. The server is normally run on louvre (the file server in the graphics laboratory). From louvre the command

```
server -f sb_file
```

is typed where sb_file has the following contents:

```
spaceball          /dev/ttyd2
```

where the device is the actual port the Spaceball is connected to. Note that the server must be restarted for each run of the BMVS.

B.2.5 Command line options Several of the default run time parameters can be altered by using command line options. The complete syntax of the command line is as follows:

```
BMVS [-f paramfile] [-fp pathfile] [-fa aircraftfile] [-fb benfile] [-n]
```

where:

-f paramfile: use specified parameter file instead of default "paramfile"

-fp pathfile: use specified path file instead of default "pathfile"

-fa aircraftfile: use specified aircraft file instead of "aircraftfile"

-fb benfile: use specified BEN file instead of "benfile"

-n: run in NTSC mode

Note that the command line options override any options specified in the paramfile.

B.3 Expert Users

This section is designed for users quite comfortable with most of the BMVS features but want to know how to exploit it to its fullest. It can be considered a mini-programmers manual because it deals with the likely modifications to the BMVS. The changes one might want to make include changing an input devices binding's or adding a new input device.

B.3.1 Changing Mouse Bindings Because the mouse communicates to the BMVS with only its three buttons, only seven functions may be accessed with the mouse. The three mouse buttons may be thought of the three bits that make up an octal digit. The right most button is used as the least significant bit. The number 0 corresponds to no buttons being pushed and should not be bound to a function. The remaining bindings can easily be changed by changing the *define* statements in the source file fullmouse.c. The default bindings are shown below:

```
#define FORWARD 2
#define BACKWARD 5
#define FASTFORWARD 99
#define TURN_RIGHT 1
#define TURN_LEFT 4
#define JUMP 98
#define PLEASE_QUIT 97
#define TURN_UP 6
#define TURN_DOWN  3
#define DO_MENU 7
```

If the desired reaction to pushing the right button is to pan the viewpoint down then the TURN_DOWN symbol must be defined to be 1 and the TURN_RIGHT symbol must be defined to a number other than 1.

B.3.2 Adding an Input Device Driver This section explains what is involved with adding a device driver, and what existing software may need to be changed due to the addition. Four input devices are currently implemented on the BMVS: mouse, keyboard, Dimension6 and Polhemus. The source code for these devices are located in the following files: fullmouse.c, keyboard.c, dim6.c and do_polhemus.c. Each driver communicates with the main driver routine in the same way, but each is quite unique internally. It is highly recommended that the user inspect the drivers for these devices and read the material below as a first step in installing a new driver.

B.3.2.1 Communication The basic line of communication between the input device and the main driver routine is the command structure (see Table 13). Each active input device is passed a pointer to a list of command structures and adds new commands to the head of the list as dictated by the inputs to the physical device. The driver then returns the pointer to the new head of the list (may be the same as the input pointer if no new action is desired). Once all the input drivers have collectively built the command list the BMVS driver traverses the list and executes the corresponding commands. It is interesting to note that the mouse, keyboard, and Polhemus devices return at most one command while the Dimension 6 may return

Table 13. Command structure type definition

```
typedef struct command_structure
  {
    enum { NIL, MOVE_ANGLE, MOVE, MOVE_ABS, GRAB,
   TILT_ANGLE,SELECT_BEN, MOD_SETTINGS,
   SELECT_PLANE, SELECT_CHECKPOINT, REMAKE_PATH,
   PAUSE, SELECT_PATH , TOGGLE_CLOCK,
   RESET_CLOCK, SET_CLOCK, DROP, JUMP_IN_OUT,
   LOOK, AUTOPATH ,MOVE_HAND, QUIT} action;

    POSITION_TYPE delta_angles;
    POSITION_TYPE eyepoint;
    POSITION_TYPE center_point;
    POSITION_TYPE delta;
    float         distance;
    float         sinTurn;
    float         cosTurn;
    int           redraw;
    TIME_TYPE     time;
    PATH_TYPE     *path_ptr;
    AIRCRAFT_TYPE *air_ptr;
    BEN_TYPE      *ben_ptr;
    CHECKPT_TYPE *ck_ptr;
    struct command_structure *next_command;
    int *hook       /* user hook for later uses */
  }COMMAND_TYPE;
```

several. For example, the Dimension 6 allows the user to move forward and turn right at the same time whereas the mouse can only do one or the other.

For efficiency reasons the driver routines are given access to a few global variables. The head pointers for the aircraft, BEN, and path lists as well as the pointers to the selected aircraft, BEN, and path objects.

B.3.2.2 Program Modifications. Normally very few modifications should be needed in the main sections of the BMVS when adding an input device. No changes should be necessary when adding either a joystick or VPL Data Glove driver. The makefile already links in the stubs joystick.c and glove.c and the settings structure has references to the devices. So all that is needed in these cases is to write the device drivers to replace the stubs.

When adding a new device other than the joystick or glove the amount of modification needed depends on the answer to one question: "can the new device operate within the existing command structure"? If the answer is yes then only part one below needs to be completed, if however, the answer is no then parts one and two must be completed to install the new driver.

Part One. The following changes must be made:

- The settings structure must be modified to include a value to indicate whether the device will be active or not (for this example call it USE_NEW). If the device uses a tty port a second variable must be added to store the port name.
- Init.c must be modified to set the default states of the variables added to the settings structure and to read in the values from a paramfile.
- The driver function is forward referenced in the header file *protos.h*.
- Make_mover.c must add tests to check USE_NEW and if true add the address to the driver function onto the mover_table. If the driver requires it, its initialization routine is then called.
- Check to see that adding this driver doesn't make the total number of drivers exceed the maximum allowed (constant MAX_DRIVERS).

Part Two. When a completely new command is added the following changes are made in addition to the ones in part one:

- Modify the command structure to accommodate the new command.
- Add a case in the switch statement in driver.c to handle the case of this new command.

B.4 File Formats

The BMVS makes use of many files when it runs. Most of the files used are AFIT format geometry files which describe the polygon descriptions of various objects. The format for the geometry files is not discussed here as it is available elsewhere. Four other files types are used by the BMVS and are described below.

B.4.1 The Parameter File The parameter file allows the user a method to set the run-time configuration of the BMVS. Display mode (NTSC vs RGB), input devices used, and whether paths are displayed or not are a few of the options that can be included in the parameter file. The format of the parameter is as follows:

A series of lines, each beginning with a keyword followed by a space and one or more corresponding parameters.

line = [keyword: value] [value ...]

Although any line not beginning with a keyword is ignored a convention of starting a comment line with a "#" symbol has been adopted. All of the paramfile parameters are optional and when a parameter is ommitted its default values will be used. See table 14 for a complete listing of all the keywords. A short example of a paramfile follows:

```
DIM6: 1
MODE: NTSC
PATH_STYLE: 2
```

This paramfile will cause the dimension 6 to be used as an input device, NTSC mode will be used, and a moderately complex path object will be shown. Note that the keyboard and mouse will also be used while the glove, joystick and Polhemus will not due to the default settings.

Table 14. Parameter File Keywords

Keyword	Values	Description
AIRCRAFTFILE:	String	To specify an alternate aircraft file
PATHFILE:	String	To specify an alternate path file
BENFILE:	String	To specify an alternate BEN file
MODE:	String	"NTSC" for NTSC mode
POLHEMUS:	Boolean	A 1 indicates the Polhemus is to be used
MOUSE:	Boolean	A 1 means use mouse
GLOVE:	Boolean	A 1 means use glove
DIM6:	Boolean	A 1 means use dimension 6 (spaceball)
JOYSTICK:	Boolean	A 1 means use joystick
BLUE_BACK:	Boolean	0 for black background, 1 for blue
CHECKPOINT_ACTIVE:	Boolean	Spline control points shown or not
PATHS_ACTIVE:	Boolean	0 for no paths, 1 shows paths
BEN_ACTIVE:	Boolean	1 to show BEN objects
SAM_ACTIVE:	Boolean	Reserved for later use
AIRPORTS_ACTIVE:	Boolean	Reserved for later use
AIRCRAFT_ACTIVE:	Boolean	Reserved for later use
TICK:	Integer	Tenths of seconds per frame (default 10)
DO_FLOOR:	Boolean	1 paints green floor, 0 for wireframe only
DIM6_PORT:	String	Tty port or "REMOTE" when using server
POLHEMUS_PORT:	String	Tty port or "REMOTE" to use server
GLOVE_PORT:	String	Tty port or "REMOTE" to use server
JOYSTICK_PORT:	String	Tty port or "REMOTE" to use server
PATH_STYLE:	Integer	Which path style to use (default 1)
XRANGE:	2 floats	Min and max X for drawing terrain grid
YRANGE.	2 floats	Min and max Y for drawing terrain grid

B.4.2 The BEN File The BEN file (so named after the Basic Encyclopedia Numbers used in ATOs to reference static objects) is used to describe the objects and locations within the environment that are not aircraft, terrain, or paths. Examples of BEN file objects are SAM sites, targets, TCAs, etc.

The BEN file consists of 1 or more object descriptions. The object descriptions begin with the keyword OBJECT and end with the keyword END. Between the OBJECT and END keywords are one or more lines that have a similar format to the paramfile.

```
OBJECT
    [keyword: value][value ...]
END
```

Like the paramfile, unspecified values in the BEN file for a given object will take on default values. See Table 15 for a list of all the keywords. A short example BEN file follows:

```
OBJECT
TYPE: SAM
NAME: SAM00X1
LOCATION: 320000 320000 0
COLOR: RED
FRIENDLY: FOE
MODELS_ACTIVE: 0 1
GFILE: tca
END
OBJECT
TYPE: TARGET
NAME: Airfield one
LOCATION: 125000 -77000
COLOR: RED
FRIENDLY: FOE
MODELS_ACTIVE: 1 0
END
```

Table 15. BEN File Keywords

Keyword	Values	Description
OBJECT		Begins an object description
END		Ends an object description
NAME:	String	Name of the BEN object
TYPE:	String	Type of BEN object e.g., airport, SAM, etc.
GFILE:	String	User specified geometry file
ACTIVE:	Boolean	0 for not displayed , 1 for displayed
LOCATION:	3 floats	Location in 3 space (x,y,z)
COLOR:	String	Object's color object e.g., "RED" "BLUE" etc.
FRIENDLY:	String	"FRIEND" "FOE" or "NEUTRAL"
FINISH:	3 floats	3 space location (complex objects only)
WIDTH:	float	(Complex objects only i.e., attack routes)
HEIGHT:	float	(Complex objects only i.e., attack routes)
MODELS_ACTIVE:	2 booleans	One for solid model shown other for wireframe

This BEN file would produce two objects, one SAM site (rendered using the geometry file "tca") and one target. The SAM site would be displayed with vectors (MODELS_ACTIVE: 0 1) while the target would be displayed with filled polygons (MODELS_ACTIVE: 1 0).

B.4.3 The Aircraft File The aircraft file is the mechanism used to specify what aircraft are flown and at what times. The format of the aircraft file is very similar to the BEN file. Each aircraft description falls between an OBJECT and END keyword.

```
OBJECT
   [keyword: value][value ...]
END
```

See Table 16 for a list of all the aircraft file keywords. An example aircraft file follows:

```
#aircraftfile
```

```
OBJECT
NAME: F15
PATH: testpath1
POSITION: 0. 0. 30.
COLOR: RED
SPEED: 420.
TOT: 09 00
#GFILE: fancy_f15
END
OBJECT
NAME: F4
PATH: testpath2
COLOR: BLUE
TOT: 11 00
END
```

This aircraftfile creates two aircraft objects (a blue F4 and a red F15). The F15 will fly at 420 MPH while the F4 will fly at a default value. The F15 will fly about 30 feet above testpath1 based on the values given with the keyword POSITION:. The graphic object rendered for the F15 will be make from the default f15 geometry file rather than the "fancy_f15" file because the GFILE line is commented out.

B.4.4 The Path File The path file allows the user to describe a path with a few control points. The path file format resembles the AFIT geometry file format and is as follows:

Line 1: comment

Line 2: number of path records to follow

path records begin at line 3

    path record format

    Line 1: pathname

    Line 2: number of control points following

    control point records begin at line 3 of a path record

        Line1: position (x, y, z)

Table 16. Aircraft File Format

Keyword	Values	Description
OBJECT		Begins an object description
END		Ends an object description
NAME:	String	Name of the aircraft object
GFILE:	String	User specified geometry file
ACTIVE:	Boolean	0 for not displayed , 1 for displayed
POSITION:	3 floats	Offset from path in 3 space (x,y,z)
COLOR:	String	Color of the object e.g., "RED", "BLUE", etc.
PATH:	String	Name of the path this plane is to follow
PACKAGE:	String	Name of the package this aircraft belongs to
MISSION:	String	Mission identifier
ORDANANCE:	String	Type of ordnance (default "SCL")
TOT:	2 integers	Hours minutes for time over target
START_TIME:	2 integers	Hours minutes for start time
SPEED:	float	The aircraft's speed

Line2: type of checkpoint (home, fuel, target, etc.)

An example pathfile follows:

```
#paths
2
testpath1
5
0 0 0
home
7000 7000 3070
fuel 40000 0 10000 path 190000 130000 33000
unknown
125000 -77000 9000
target
0 0 0
home
testpath2
5
0 0 0
```

```
home
-25000 -55000 10000
fuel
-500000 -500000 5000
target
500000 -500000 45000
unknown
0 0 0
home
```

This pathfile creates 2 paths (testpath1 and testpath2) each with 5 control points used to create the curved path the aircraft will fly.

Appendix C. *Unix Manual Page*

```
BMVS(1)                 USER COMMANDS                 BMVS(1)


NAME
     BMVS - preview the execution of an Air Tasking Order

SYNOPSIS
     BMVS [ -f paramfile ] [ -fb benfile ] [ -fa aircraftfile ]
          [ -fp pathfile ] [ -n ]

DESCRIPTION
     BMVS is a time dependent simulation  of  an  ATO  execution.
     The  user  can  change  viewpoints within the environment or
     "fly" along an aircraft's path.  The user  may  also  modify
     the  environment  by  moving  objects such as targets to new
     locations.  The BMVS will supports the use  of  mouse,  key-
     board,  Spaceball,  and  Polhemus Tracker input devices.  By
     default BMVS comes up with the mouse, and  keyboard  active.
     The  contents of the paramfile can alter which input devices
     are active.


OPTIONS
     -f paramfile
          Use the specified paramfile  rather  than  the  default
          "paramfile".

     -fb benfile
          Use the specified benfile rather than the default "ben-
          file".

     -fa aircraftfile
          Use the specified aircraftfile rather than the  default
          "aircraftfile".
```

-fp pathfile
    Use the specified pathfile rather than the default "pathfile".

-n Run in NTSC mode.

FILES

./paramfile	runtime configuration
./benfile	specifies location of BEN objects
./aircraftfile	specifies the aircraft used in this simulation
./pathfile	specifies the paths the aircraft will follow
./RangeData	geometry file
./bigbox	geometry file
./f15, ./f16 , ./f5	geometry files
./f4, ./f111, ./a10	geometry files

SEE ALSO
    server

BUGS
    The formats for the required files must be strictly followed. Variations will cause unexpected results.

Bibliography

1. Booch, Grady. *Software Engineering With Ada.* Menlo Park, CA: Benjamin/Cummings, 1983.
2. Buchanan, Maj Thomas H. "The Need For Battle Managers in the TACS," *Air Power Journal*, pages 61–71 (Summer 1987).
3. Filer, Robert. Personal interview. AFIT/ENG, June 1989.
4. Filer, Robert. *A Virtural Environment Display System.* MS thesis, Air Force Institute of Technology, Wright-Patterson AFB, 1989.
5. Foley, James D. and Andries Van Dam. *Fundamentals of Interactive Computer Graphics.* Reading MA: Addison-Wesley, 1982.
6. G. Frekany and others. *Functional Description(Final) Tactical Expert Mission Planner (TEMPLAR).* Contract F30602-85-C-0249, TRW Defense Systems Group, March 1987.
7. Harris, Frank E. and others., "Preliminary Work on the Command and Control Workstation of the Future: Progress Report." O&MN Direct Funding. San Diego, October 1987 - September 1988. CA: Naval Postgraduate School, August 1988.
8. Horgan, John. "Inside the Black Box," *IEEE Spectrum*, 23.2+ (November 1986).
9. Kanko, Capt Mark A. and Maj Phil Amburn. "Geometric Modeling of Flight Information for Graphical Cockpit Display." In *National Areospace Electronics Conference*, pages 201–208, New York: IEEE Publishing Services, 1988.
10. Lorimor, Gary and others. "No More 9-G Wrists." Unpublished paper. Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 9 January 1989.
11. Lorimor, Gary K. *Real-Time Display of Time Dependent Data Using a Head-Mounted Display.* MS thesis, AFIT/GE/ENG/88D-22. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, 1988.
12. "NASA's Virtual Workstation: Using Computers to Alter Reality," *NASA Tech Briefs*, pages 20–21 (July/August 1988).
13. Smith, Alvy Ray, "Spline Tutorial Notes." Technical Memo Lucasfilm Ltd., May 1983.
14. Sutherland, Ivan E. "A Head-Mounted Three Dimensional Display." In *Proceedings of the AFIPS Fall Joint Computer Conference*, pages 757–764, 1968.
15. Thomas, Ivan and others. *Advanced Tactical Air Control Center Functional Description.* Sperry Corporation, St Paul, MN, November 1986. Contract #F30602-84-C-0117.

16. VPL Research, Inc. *Eyephone System*. Product Announcement. Redwood City CA, July 1989.

17. Wohl, Joseph G. "Force Management Decision Requirements for Air Force Tactical Command and Control," *IEEE Transactions on Systems, Man, and Cybernetics*, *11*:618–639 (September 1981).

Vita

Captain Charles L. Wardin [redacted] In 1977 He graduated from Colorado Mountain College in Glenwood Springs, Colorado with an associates degree in photography. He then worked as a professional photographer until 1980 when he joined the Air Force. His first assignment was at North American Aerospace Defense Command's (NORAD) Cheyenne Mountain Complex as a computer programmer. From 1982 to 1985 he attended Colorado University as part of the Airmans Education Commissioning Program. After receiving a Bachelor of Science in electrical engineering, he atended Officer Training School. From there he went to Electronic Systems Division, Hanscom AFB, Massachusetts. He worked in the Command Center Processing and Display System - Replacement program office at Hanscom until entering The Air Force Institute of Technology in 1988. [redacted]

[redacted]

[redacted]

· UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/ENG/89D-56	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering — 6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology (AU) Wright-Patterson AFB, Ohio 45433-6583	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING / SPONSORING ORGANIZATION AAMRL — 8b. OFFICE SYMBOL (If applicable) HEA	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) Wright-Patterson AFB, Ohio 45433-6583	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
BATTLE MANAGEMENT VISUALIZATION SYSTEM

12. PERSONAL AUTHOR(S) Charles L. Wardin, Capt, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS Thesis	FROM ______ TO ______	1989 December	80

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
25	05		Tactical data systems; Helmet mounted displays
12	05		Computer Graphics; Command and control systems

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Thesis Advisor: Phil Amburn , Major, USAF
Professor of computer systems

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Phil Amburn, Professor of computer systems	22b. TELEPHONE (Include Area Code) (513) 225-4279 — 22c. OFFICE SYMBOL ENG

DD Form 1473, JUN 86   Previous editions are obsolete   SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

UNCLASSIFIED

This investigation deals with the development of a software system to explore the concept of previewing Air Tasking Orders in a three-dimensional virtual environment. The virtual environment is created by displaying three-dimensional graphics images onto a graphics workstation screen or onto a head-mounted display. The software system, entitled the Battle Management Visualization System (BMVS), supports a variety of input devices to suit different situations. The user is presented with a miniature battle environment complete with aircraft, threat regions, targets, etc. The aircraft *fly* along paths and arrive over the targets at the simulated time over target time. The user may also *fly* along the path to create a plane's eye view. The user can *move* about the environment to examine various aspects close up or zoom out for an overall view. The results of the effort support the idea that a tool to preview ATOs is feasible.

UNCLASSIFIED